CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Lancaster E-Prints

Revisiting Ontology-Based Requirements Engineering in the age of the Semantic Web

Glen Dobson, Peter Sawyer
Computing Department, Lancaster University
{g.dobson, p.sawyer}@lancs.ac.uk

Abstract

There is a long history of research into utilising ontologies in the Requirements Engineering process. An ontology is generally based upon some logical formalism, and has the benefits for requirements of explicitly modelling domain knowledge in a machine interpretable way, e.g. allowing requirements to be traced and checked for consistency by an inference engine, and software specifications to be derived.

With the emergence of the semantic web, the interest in ontologies for Requirements Engineering is on the increase. Whilst efforts have been concentrated upon re-interpreting software engineering techniques for the semantic web, it is interesting to consider what benefits there are to be passed from the semantic web to traditional Software Engineering techniques.

In this paper we give an overview of this emerging research field, suggesting directions that could usefully be taken in the field of dependability requirements. We present our work on a dependability ontology compliant with the IFIP Working Group 10.4 taxonomy and discuss how this, and other ontologies, must interact in the course of Dependability Requirements Engineering. In particular we consider the links between the dependability ontology, an ontology for requirements and domain ontologies, identifying the advantages and difficulties of this approach.

1. INTRODUCTION

An ontology as the term is used in the field of knowledge representation is most often defined as “a representation of a conceptualization” [1]. A more detailed description of an ontology is that it is a formal representation of the entities and relationships which exist in some domain. It should also represent a *shared* conceptualisation in order to have any useful purpose.

There are clear overlaps between what an Ontology Engineer aims to achieve in the modelling of a domain and the modelling that a Requirements Engineer will perform during the requirements process. Ontologies offer one possibility for representing, organising and reasoning over the complex sets of knowledge that requirement documents embody. Because of the synergy between the two fields of research, numerous works dating back at least two decades have addressed the use of ontologies in Requirements Engineering; e.g. [2], [3], [4], [5], [6]. More widely speaking, all formalisms for Requirements Engineering embody a particular conceptualisation, and many (probably most) are reducible to first order logic. Therefore, even these other formalisms have much in common with ontologies.

Since these works were published there has been a renewed interest in ontologies due to the emergence of the semantic web. As with the Requirements Engineering research mentioned above, the semantic web builds upon earlier work on knowledge representation. The emphasis now is on sharing ontologies via the web, and machine inference over heterogeneous data sets. There is an increasing amount of research devoted to utilising semantic web technologies in

software engineering, and Requirements Engineering in particular (e.g. [6], [7]). In this paper one of our aims is simply to draw attention to and further examine this trend.

Whilst much work has concentrated upon ontologies representing requirements models (and metamodels), little effort has been made to address specific areas such as Dependability Requirements Engineering. Since dependability is such an important sub-area for many systems, and involves a complex set of concepts of its own we see this as a shortcoming. Therefore our primary aim in this paper is to examine what might be required of an ontology-based dependability process. Clearly, a dependability ontology is central to enabling such a process. In this light, we discuss a dependability ontology, which has emerged from our own research conducted as part of the EU-funded SeCSE (Service-Centric System Engineering) Integrated Project [9].

The remainder of the paper is structured as follows: In Section 1 we revisit the concept of ontologies for Requirements Engineering, discuss the various forms that this may take and the advantages of the approach (see Section 2). In Section 3 we look at ontologies in the semantic web, how and why semantic web technologies might be applied to requirements, as well as how these technologies compare to their predecessors. In Section 4 we look at ontology-based Dependability Requirements Engineering in the age of the semantic web. Finally, in Section 5, we draw conclusions on the applicability of semantic web technology in this domain and suggest future research directions.

2. THE USE OF ONTOLOGIES IN REQUIREMENTS ENGINEERING

The emphasis in an ontology language is on providing a formality for representing knowledge in such a way that deductive inferences can be drawn by a machine. This almost universally means that the formalism is based upon first order logic.

Ontologies are useful for representing and interrelating many types of knowledge. The nature of requirements engineering involves capturing knowledge from many sources. There are therefore many potential uses of ontologies in requirements engineering including the representation of:

- The requirements model itself, imposing and enabling a particular paradigmatic way of structuring requirements
- Acquisition structures for domain knowledge
- The application domain
- The environment

Most of the above are generally enabled by an ontology (or set of ontologies) which use the same underlying “ontology language”. This makes it easier to interrelate knowledge ion different areas and offers a unified, underlying conceptualisation to the requirements process. As well as those mentioned above, ontologies can also be used to represent other reusable models that are relevant to requirements. This is discussed for the example of dependability in Section 4. Figure 1, in which the arrows represent conceptual dependencies depicts the interrelations between different ontologies in requirements engineering.

2.1 Existing Work on Ontology-based Requirements

Much of the groundwork for ontology-based Requirements Engineering was laid during the development of the RML (Requirements Modelling Language) framework [2] in the early 1980s. RML was built upon the following broad principles [10]:

- There is more to writing requirements than functional specification
- Requirements should be developed and presented as models
- Given the previous two points, it is *conceptual models* that should be developed
- Abstraction and refinement, especially involving Is-A hierarchies, are significant in engineering large requirements
- Formal requirements modelling languages are needed

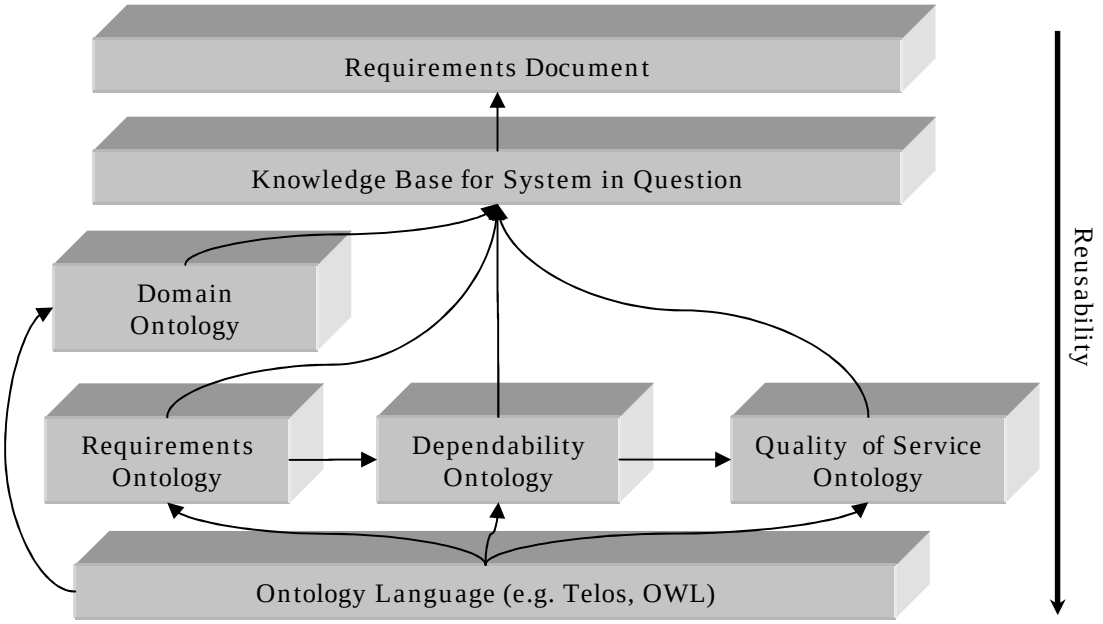

Figure 1. Ontologies in Requirements Engineering

In RML, models are built of *individuals* grouped into *classes*, which are in turn instances of *metaclasses*. *Classes* and *metaclasses* have *definitional properties*. *Individuals* have *factual properties* which specify instance information for the *class*' *definitional properties*. In essence, RML defined its own ontology language. It will become clear in the next section just how close the building blocks of RML are to those used in the semantic web's ontology language. This is not surprising as they share a common ancestor in semantic networks [11]. Using this "ontology language" RML provided an ontology for requirements modelling. This centred around three metaclasses: *Entity*, *Activity* and *Assertion*.

Whilst these concepts are quite generic, this did mean that RML modelled only a single paradigm for requirements expression. Telos [3], was a language which evolved from RML which enabled extensible requirements ontologies (e.g. allowing the concepts of Agents, Goals, Actors, Tasks, etc. to be added to the model as appropriate to the paradigm). Amongst other improvements Telos made over RML it included support for time intervals and temporal relations allowing the history of a domain to be captured. The O-Telos dialect of Telos is still in use today thanks to the success of the ConceptBase [12] implementation of the language, and has also come to be used as a general purpose knowledge representation language.

Kaos [4] which emerged soon after Telos also supports multiple paradigms through a generic ontology which forms a metamodel for requirements. As well as a conceptual layer much like Telos, Kaos separated out the temporal aspects of the language and underpinned them with a semantics based upon first order temporal logic. Albert II [5] also emerged around the same time. This language again concentrates upon the application of first order temporal logic, but can perhaps be put into historical context as a Telos ontology as it depends directly upon ConceptBase (and thus the Telos knowledge representation language). The ontology it defines concentrates upon the Agent concept, structuring it in terms of Declarations of State Components and Actions and logical Constraints allowing an agent to be classified.

The i* framework [6] also builds upon the foundations of RML and Telos, but concentrates upon the modelling of the wider business process. To do so it introduces two main components – the Actor Dependency model, which models organisations as a network of interdependent actors; and the Issue Argumentation model, which captures arguments about the relative merits of alternative designs. The concept of a "soft-goal" is also supported, allowing non-functional requirements which have no sharp definition to be defined (among other things).

More recently work has emerged which addresses more or less the same area as we propose here. [13] defines a common language for requirements engineering (a very similar aim to Telos), which is in fact ontology-based. The ontological framework used is also generic enough to support representation using semantic web technologies. The common language combines the goals, scenarios and viewpoint paradigms (but like Telos other paradigms could presumably be supported using the underlying framework). This work does also address the specific area of Dependability Requirements Engineering – but concentrates upon security rather than dependability in the traditional sense. We believe that there are still important issues to be addressed in the adaptation of ontology-based Requirements Engineering techniques to Dependability Requirements Engineering, and that many of these centre on the definition of an ontology for dependability (including security). Section 4 begins to look at this. Specifically, it is important to have integrated requirements (and therefore ontologies) because dependability, as with other aspects of a system, can only be fully understood in context.

3. ONTOLOGIES IN THE AGE OF THE SEMANTIC WEB

The semantic web is a movement to make the semantics of web-content accessible to machines. It has been summarised by its originators as "*an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*" [14]. Berners-Lee has since emphasized that the semantic web is not about adding semantics to existing HTML content, but about enabling machine inferences across multiple datasets (e.g. existing data in relational databases or XML format). The web involved is no longer a web of documents but of data. With the added "machine understanding" of data, many tasks that previously had to be performed by human agents can be performed by software agents.

The RDF (Resource Description Framework) [15] layer of the semantic web builds upon XML (eXtensible Markup Language). RDF is a language for representing information about resources in the World Wide Web in a way that is suitable for processing by applications. All resources are identified by URI, and the basic structure of the language consists of statements of the form subject - predicate – object. For instance in the following statement the subject is

the URL http://www.example.org/index.html, the predicate is the word "creator" and the object is the phrase "John Smith":

http://www.example.org/index.html has a *creator* whose value is *John Smith*

An XML syntax is defined for interchange of RDF. The RDF framework also consists of a vocabulary description language – RDF Schema (RDFS). This allows for the definition of Classes of resource in terms of their properties and by stating subclass relationships with existing classes. Whilst RDFS can be viewed as a simple ontology language, the semantic web stack contains a further layer on top of RDF – the ontology layer.

3.1 The Web Ontology Language (OWL)

OWL (the Web Ontology Language) [16] has come to dominate the ontology layer of the semantic web and in context of this paper can be considered to be analogous to e.g. Telos. OWL is an RDF (and therefore also XML) language. However, behind the RDF/XML syntax OWL is really a Description Logic (DL) [17]. Indeed the most useful "species" of OWL is named OWL-DL for this reason. A Description Logic is a logic that focuses on concept descriptions as a means of knowledge representation and has semantics which can be translated to first-order predicate logic. The nature of DLs means that classification, subsumption and satisfiability can be automatically computed by a reasoner. OWL can be seen as a trade-off between expressivity and decidability. In this context decidable means that inference algorithms exist for the language and are known to terminate. Of the three "species" of OWL, OWL-DL and its subset OWL-Lite are decidable while OWL-Full is not. Already, one can begin to see potential advantages to utilising this new generation of ontology language for requirements. In [10] it is stated that "*The field of Requirements Engineering like knowledge representation, must eventually come to terms with the computational intractability (even undecidability) of reasoning with most expressive logical formalisms: if we are to have useful tools, we cannot allows them to unexpectedly go off into 'trances'*". The decidability and complexity properties of OWL-DL are very well understood compared to earlier knowledge representation techniques.

In DL reasoning, an open world assumption is made. This means that things that are not explicitly asserted are taken to be unknown. This contrasts with the closed world assumption widely used in data modelling, where anything unstated is taken to be false. For instance, if an Activity X is stated to be performed by an Agent Y then, in the absence of any other knowledge, asking "is Activity X performed by agent Z?" will result in the answer "unknown" in an open world, but the answer "no" in a closed world.

An OWL ontology consists of Classes and their Properties. Instances of OWL Classes are called Individuals. As mentioned earlier, there are strong echoes of the RML/Telos ontology language in these basic language elements. Metaclasses are also supported in OWL (in the form of a Class which has Classes as its members) – but only in OWL-Full as this is a feature which leads to undecidability.

OWL Individuals are very much like resources described using RDF although they may have further OWL-specific facts expressed about them. An OWL Class is a specialization of RDFS Class, which can be specified in new ways beyond simply stating its name. These added means of class description are:

- Enumeration of all Class members (i.e. OWL Individuals) using the OWL oneOf construct.

- As an anonymous Class of all Individuals that satisfy a Property restriction using the various OWL value and cardinality constraint constructs: allValuesFrom (∀), someValuesFrom (∃), hasValue, maxCardinality, minCardinality, cardinality
- By combining existing Classes using set operators (the OWL intersectionOf (⊓), unionOf (⊔), and complementOf (¬) constructs)

These Class descriptions can be nested to create arbitrarily complex new descriptions. Descriptions can then be combined into a Class definition using the OWL subClassOf (⊑), equivalentClass (≡) and disjointWith constructs. The Class definition specifies all of the necessary and/or sufficient conditions for Individuals to be members of a Class. A Class can therefore be viewed as defining a set of individuals (the class extension).

On first using OWL, it quickly becomes obvious that there are various things it is hard to express. Thankfully these are mainly addressed by rules languages such as SWRL (the Semantic Web Rules Language) [18]. This adds the ability to express rules of the form A → B, where A and B are sets of atoms which might include (among other things) Classes or Properties. At the same time SWRL adds built-ins which allow arithmetic expressions to be used, which OWL on its own does not. Unfortunately, whilst SWRL extends OWL with a more expressive formalism it is undecidable (i.e. SWRL inference is not guaranteed to always terminate). To at least match the expressivity of the likes of Telos, SWRL's Horn-like rules are necessary. Thankfully, in some situations at least, the "DL-safe" subset of SWRL identified in [19] may be sufficiently expressive – but the important point is that we are able to separate out the decidable from the undecidable. Since this is possible, one can gain the full benefits of machine inference where necessary, whilst also gaining the full benefits of expressivity (probably at the expense of inference) where necessary.

3.2 OWL and SWRL for Requirements Engineering ontologies

Section 2.1 looked at the history of ontologies in Requirements Engineering. Despite the quantity of work in this area (by no means all of which is represented here), it does seem that much of this work is essentially about using a Telos-like formalism to provide a requirements ontology. The innovation is generally not in the underlying formalism. Giving the degree of similarity between the ontology language of OWL (including SWRL) and that used in Telos-like knowledge-based requirements techniques it is worth questioning what, if anything, can be gained by reinterpreting this work using the newer technologies.

The previous section discussed the main technical advantage – that decidability and complexity of the current generation of languages is better understood. This means that the danger of supporting software tools having unpredictable performance and in particular the risk of non-termination can be avoided. Other than this the reasons for using semantic web technologies for requirements are more pragmatic.

Firstly, the widespread success of the semantic web would mean the availability of a large number of ontologies. Furthermore, the potential for reuse of other conceptualisations through cross-referencing is enhanced in the semantic web due to the ubiquity of the web infrastructure. Together these factors would mean that a Requirements Engineer would be much more likely to find numerous reusable ontologies, which are compatible with each other (i.e. the relationships between their concepts are defined where relevant). This is particularly likely for domain ontologies, and indeed these are already beginning to emerge. As well as domain ontologies a Requirements Engineer could also expect to find reusable ontologies for

particular architectural styles (e.g. the OWL-S ontology is already a potentially useful cornerstone for modelling service-based systems), quality characteristics, etc.

The level of tool and API support for semantic web technologies is also very good, and improving. This may even prove to be the most important motivation for a semantic web based Requirements Engineering framework. The difficulty in understanding knowledge-based techniques is a well-known issue and this is only made worse in Requirements Engineering since requirements documents are aimed at a range of stakeholders from different backgrounds and domains of knowledge. Improved tool support in terms of addressing HCI issues is the way that this can best be addressed.

4. ONTOLOGY-BASED DEPENDABILITY REQUIREMENTS ENGINEERING

Part of our own work has involved the definition of an OWL dependability ontology. In the following section we discuss the conceptualisation on which this is based. The subsequent section looks at the ontology itself.

4.1 Towards a Unified Dependability Conceptualisation

A unified conceptualisation of dependability is an area which has seen ongoing work since the early 1980s. This has chiefly been undertaken by the IEEE Computer Society Technical Committee on Fault Tolerant Computing and IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. An up-to-date discussion of this work is given in [20]. For the sake of brevity, this work will be referred to as IFIP in the remainder of this section.

UMD (Unified Model of Dependability) [21] is another important development in this area. As well as providing a conceptual model of dependability as its name suggests, UMD also provides a structured way for eliciting and organising both functional and non-functional dependability requirements. UMD appears to complement the IFIP dependability conceptualisation rather than replace or supersede it. In fact, rather than provide a fixed definition of dependability UMD aims to provide a ”common language” that others can use to communicate and understand dependability despite their (potentially) different definitions. UMD could therefore be viewed as a dependability metamodel, whilst IFIP is perhaps the most authoritative dependability model. It is therefore our view that an ontology of dependability must seek to be compatible with these two works whilst providing a machine interpretable representation of the conceptualisations they embody.

IFIP define the attributes of dependability as:

- Availability: readiness for correct service
- Reliability: continuity of correct service
- Safety: absence of catastrophic consequences on the user(s) and the environment
- Integrity: absence of improper system alterations
- Maintainability: ability to undergo modifications and repairs
- Confidentiality: the absence of unauthorised disclosure of information

The dependability requirements specification for a system should include the requirements for these attributes in terms of acceptable frequency and severity of service failures for specified classes of faults and a given use environment. Despite concentrating on the above attributes IFIP do note that other system properties have an effect on dependability – but that the

definition of these is beyond their scope. This short list also hides the fact that they define secondary attributes which in some way specialise primary attributes (or combinations of primary attributes), e.g. the secondary attribute robustness is the dependability with respect to *external* faults.

UMD seeks to accommodate arbitrary attributes and their respective definitions. It should therefore be fully compatible with those listed above. Indeed, [21] demonstrates this both for IFIP and other sources of definitions of dependability attributes. The key underlying concept in UMD which allows this integration of many definitions is that of an issue. A central claim made by UMD is that all dependability attributes can be expressed in terms of issues. An issue can be either a failure or a hazard. Issues are also scoped in UMD, to indicate that they can affect the whole system, or just part of it (e.g. a particular service or component). UMD also recognises that the type of event that cause an issue are important, e.g. for security it is important to distinguish an attack, for maintainability it is important to distinguish a system update, etc. There is a large degree of overlap between an event in UMD and a fault in IFIP (i.e. the adjudged or hypothesised cause of incorrect service, which leads to a failure). In fact, it appears that they are one in the same concept. IFIP specifies a detailed fault classification scheme and therefore we suggest that this could be reused for the purposes of defining what UMD calls events. For instance, in [21] of types of event include "update" which in IFIP terms is an operational, external, human-made, non-malicious, deliberate fault (and may be judged to be an incompetence fault). These translations seem possible for all of the UMD event examples given.

Both UMD and IFIP essentially define a failure as the departure of a system from *correct behaviour*. Exactly how *correct behaviour* is defined in the two works varies however. Correct behaviour is judged by users' expectations in UMD and by adherence to a functional specification in IFIP. Clearly the latter cannot be applied where there is no explicit system specification, whilst the former may be ambiguous in the absence of a specification on which users can base their expectations. Given that failure is one of the fixed concepts in the UMD this begins to show that a dependability ontology could represent an even more generic conceptualisation by, for instance, allowing both of these definitions (and perhaps others) of correct behaviour, and by modelling their relationship.

Hazards on the other hand are not covered directly by the IFIP conceptualisation. In UMD a hazard is defined as a state of a system that can lead to catastrophic consequences for the user(s) and the environment. A hazard may or may not also be a failure. It is true to say that the IFIP model does include the concept which is equivalent to the overlap of UMD's hazard and failure. That is, a failure in IFIP's definition can be characterised according to a number of viewpoints including the consequences of the failure. Any catastrophic failure in IFIP terms is therefore also a hazard in UMD terms. Again, there is clearly room for a generic conceptualisation to accommodate both of these.

UMD classifies failures according to their *type*. This means that where a failure is clearly specifically failure of a particular attribute this can be indicated. Thus there are accuracy and performance failures, but not, e.g. reliability and availability failures. The latter failures are simply classified as "other failures". Failures can also be classified, in UMD, by their impact upon availability (e.g. stopping, non-stopping). These failure classifications overlap with both the failure domain and classification viewpoint from IFIP. IFIP also provide a more structured and complete set of classification viewpoints which also include the aforementioned failure consequences, as well as failure consistency and detectability.

UMD aims to be able to express any reasonable definition of a dependability attribute using these basic constructs (events, and issues - along with their scope and their classification). For instance, taking the IFIP definition of safety from above, this can be re-expressed as "*the index of the hazards(ISSUE) created by the system or a service (SCOPE)*"; reliability could be expressed as "the *index of all failures (ISSUE) affecting the system or a service (SCOPE)*"; maintainability could be expressed as "*the index of all failures (ISSUE) affecting the system or a service (SCOPE) due to upgrades (EVENT)*". As well as permitting translation and communication between stakeholders with different dependability viewpoints, it is also argued that UMD transfers stakeholder focus away from the abstract ill-defined level of attributes (e.g. I want a reliable system) to the more concrete failure, hazard, event and scope concepts.

4.2 A Dependability Ontology – "Representation of the Conceptualisation"

Based upon the foregoing discussion of the conceptual models of IFIP and UMD, we draw the following conclusions with regards to designing an ontology representing a unified conceptualisation. We believe that UMD generalises the model given in IFIP in a useful way, particularly with regards to the use of the issue abstraction and the related breakdown of its scope and type. However, we believe that whilst event appears to be a more abstract concept than fault, in this context they are actually identical. Moreover the IFIP fault classification scheme adds another useful dimension to a unified dependability model. In other areas, the conceptual modelling of "means of dependability" from IFIP is absent from UMD and therefore the IFIP one is included unaltered. We feel that further work on the conceptualisation in this area might result in interesting possibilities with regards to suggesting both development techniques and design alternatives based upon dependability requirements. An outline of a unified ontology in terms of the Class hierarchy and some of the Properties involved is shown in Figure 2.

The failure classification scheme in IFIP is more structured and detailed than that of UMD, but does not necessarily facilitate the translation between definitions that UMD does. We therefore suggest that the set of failure classification properties be merged in the ontology and that integrity rules be imposed (using SWRL) to maintain both versions of the classification. Such rules would take the form:

$$hasImpactOnAvailability(F, stoppingFailure) \leftrightarrow hasFailureDomain(F, haltFailure),$$

where F is the OWL Class Failure. In practice this would consist of two rules one to facilitate translation of the fact stated in UMD to IFIP terminology, and one to do the reverse. There are numerous situations where complete translations from IFIP failure classifications to UMD ones is not possible.

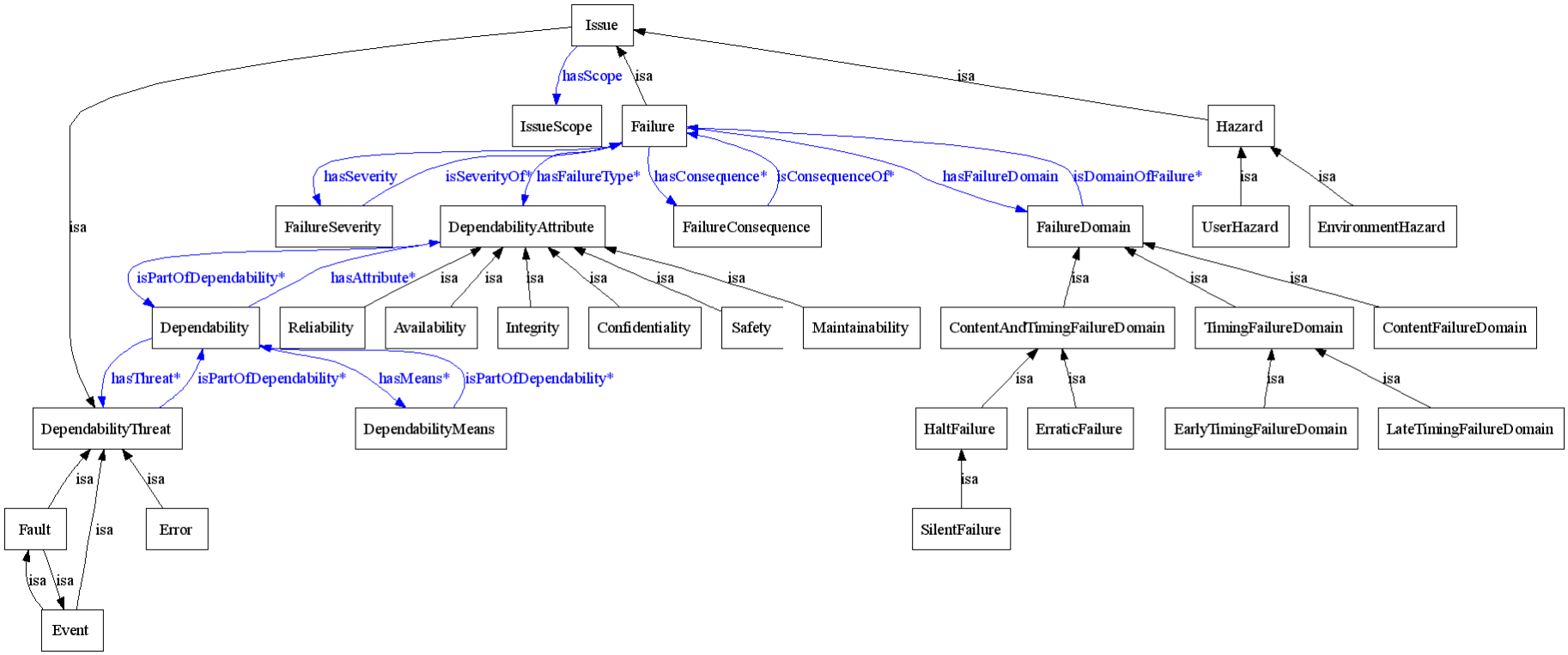

Figure 2. Overview of the dependability ontology

5. CONCLUSION

In this paper we have identified a number of research fields which appear to overlap to a large extent and which have the potential for a synergistic relationship. The field of knowledge-based requirements engineering has a long history, but it appears could benefit from the use of emerging semantic web technologies. Meanwhile the work on a conceptualisation of dependable computing also dates back a number of decades, and it seems could benefit by being made formally explicit using the same knowledge-based techniques. The resulting explicit dependability ontology could then be used along with the requirements ontologies emerging from these other fields to form the basis of a dependability requirements engineering process with strong tool support.

We believe that, unlike many other domains, in the field of dependability, thanks to the IFIP and UMD models, a real consensus seems to exist. The use of a dependability ontology therefore both serves the need for rigour that is special to Dependability Requirements Engineering and benefits from a defined scope that is tractable to reasoning.

This paper has merely set the scene for the use of ontologies in Dependability Requirements Engineering. For fuller details on the motivations behind using machine inference in Requirements Engineering one can revisit the works referenced herein (e.g. [2], [3], [7]). In the future, we hope to demonstrate these advantages by building a set of tools to enable an integrated approach to ontology-based Dependability Requirements Engineering, and to demonstrate the use of these tools in the engineering of non-trivial systems.

6. ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of the EU through the Service Centric Systems Engineering (SeCSE) EU Integrated Project 511680.

7. REFERENCES

[1] T.R. Gruber, “Toward principles for the design of ontologies used for knowledge sharing”, International Journal of Human and Computer Studies, Vol. 43, pp. 907-928, 1995.

[2] S. Greenspan, “A Knowledge Representation Approach to Requirements Definition”, PhD thesis, Department of Computer Science, University of Toronto, 1984.

[3] J. Mylopoulos, A. Borgida, M. Jarve, M. Koubarakis, “ Telos: Representing Knowledge About Information Systems”, ACM Transactions on Information Systems, 1990

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-Directed Requirements Acquisition”, Science of Computer Programming Vol. 20, pp.3-50, 1993.

[5] P. Du Bois, “The Albert II Language: On the Design and the Use of a Formal Specification Language for Requirements Analysis”, PhD thesis, Computer Science Department, University of Namur, Namur (Belgique), 1995.

[6] E. Yu, and J. Mylopoulos, “Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering”, in Proc. 27th Hawaii International Conference on System Sciences, Vol. 4, pp. 243-243, 1994.

[7] V. Mayank, N. Kositsyna, and M.A. Austin, "Requirements Engineering and the Semantic Web: Part II. Representation, Management and Validation of Requirements and System-Level Architectures," ISR Technical Report 2004-14, University of Maryland, 2004

[8] K. Breitman, J. Leite, “Ontology as a Requirements Engineering Product”, in Proc. 11th International Symposium on Requirements Engineering (RE ’03), pp. 309- 319, 2003

[9] SeCSE - EU Integrated Project, http://secse.eng.it

[10] S. Greenspan, J. Mylopoulos, and A. Borgida, “On Formal Requirements Modeling Languages: RML Revisited”, Proc. 16th International Conference on Software Engineering, 1994.

[11] R. Quillian, “Semantic Memory”, *Semantic Information Processing*, M.Minsky (ed.) MIT Press, pp. 220-270, 1976.

[12] M. Jarke, R. Gallersdörfer, M. A. Jeusfeld, M. Staudt, and S. Eherer, “ConceptBase — A deductive object base for meta data management”, Journal of Intelligent Information Systems, Vol. 4, No.2, pp.167-192, 1995.

[13] S-W. Lee, R. Gandhi, “Engineering Dependability Requirements for Software-intensive Systems through the Definition of a Common Language”, in Proc. 13th IEEE International Requirements Engineering Conference, Workshop on Requirements Engineering for High-Availability Systems (RHAS), pp. 40-48, 2005.

[14] Tim Berners-Lee, James Hendler, Ora Lassila, “The Semantic Web”, Scientific American, May 2001.

[15] Frank Manola, Eric Miller (ed.), “RDF Primer”, W3C Recommendation, http://www.w3.org/TR/rdf-primer/, 2004.

[16] Deborah L. McGuinness, Frank van Harmelen (ed.), “OWL Web Ontology Language Overview”, W3C Recommendation, http://www.w3.org/TR/owl-features/, 2004.

[17] F. Baader, I. Horrocks, and Ulrike Sattler, “Description logics for the semantic web”, Künstliche Intelligenz, Vol. 16, No. 4, pp. 57-59, 2002.

[18] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean. “SWRL: A Semantic Web Rule Language Combining OWL and RuleML”, W3C Member Submission. http://www.w3.org/Submission/SWRL/.

[19] B. Motik, U. Sattler, and R. Studer, “Query Answering for OWL-DL with Rules”, Proc. 3rd International Semantic Web Conference (ISWC 2004), pp. 549-563, 2004.

[20] A. Avizienis, J-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11-33, 2004.

[21] V. Basili, P. Donzelli, and S. Asgari, “A Unified Model of Dependability: Capturing Dependability in Context”, in IEEE Software, Vol. 21, No. 6, pp. 19-25, 2004.